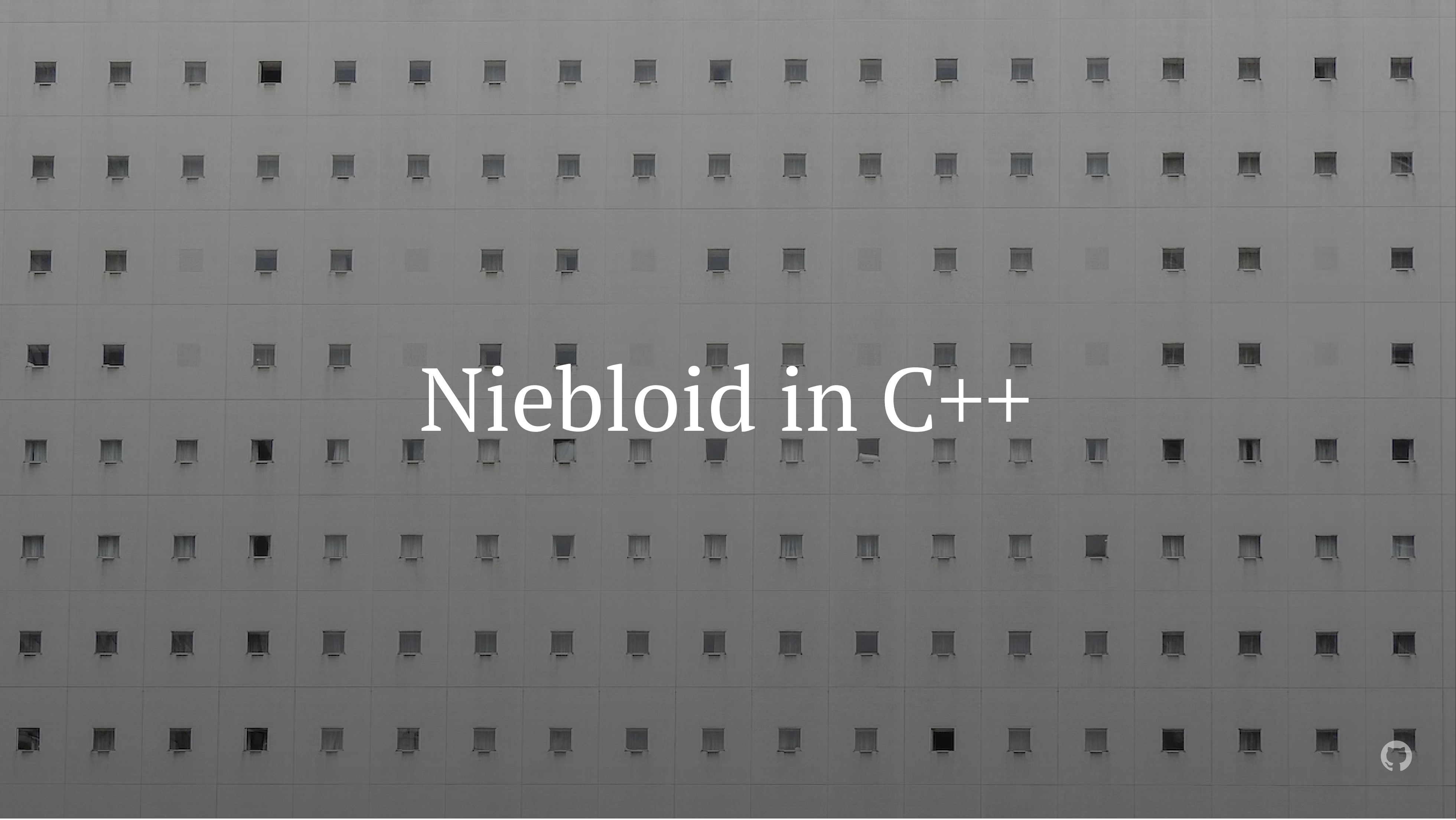


# Niebloid in C++

## Background: Function Overloading

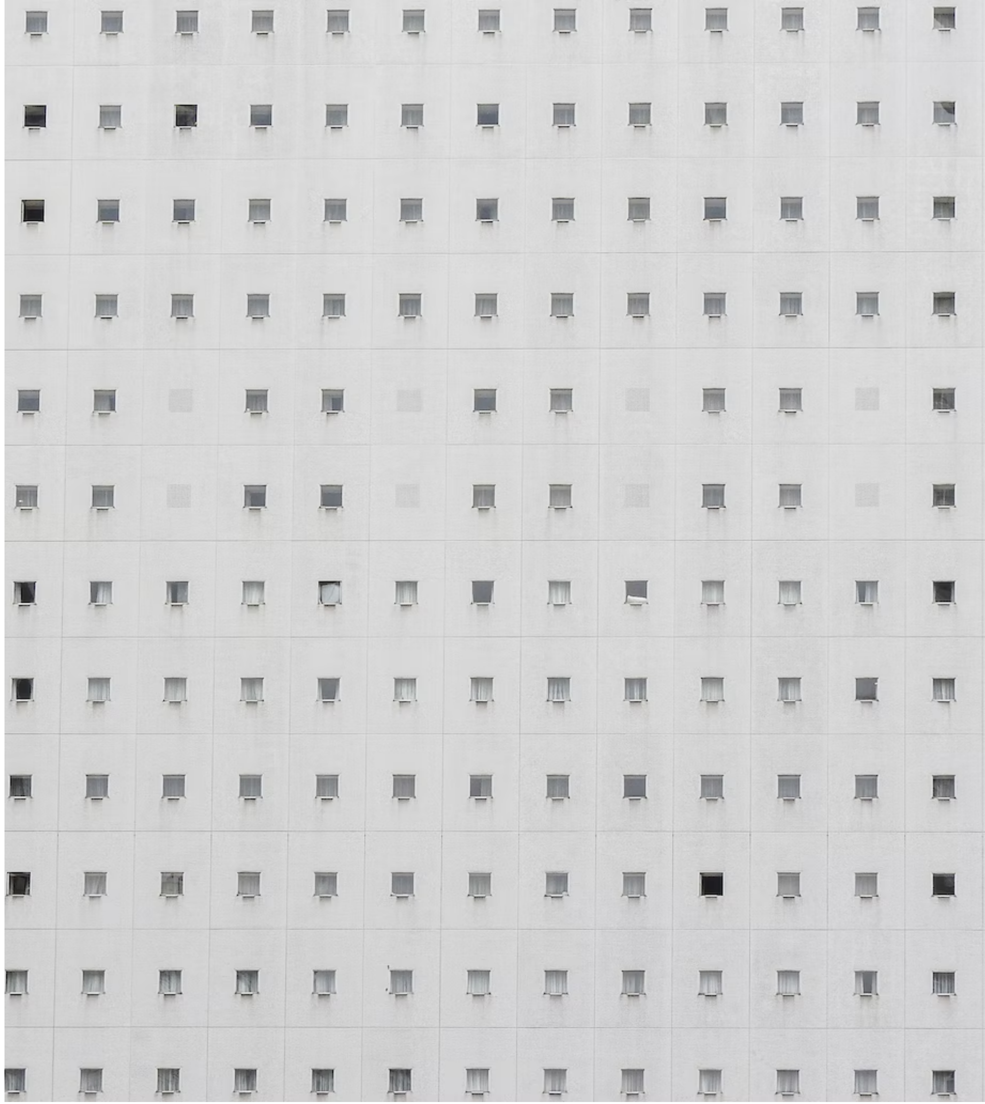
```
int a = 0;
int a = 0; // error here, can not have same name
```

```
int add(int a, int b) {
    return a + b;
}

double add(double a, double b) {
    return a + b;
}

int add(int a, int b, int c) {
    return a + b + c;
}
```

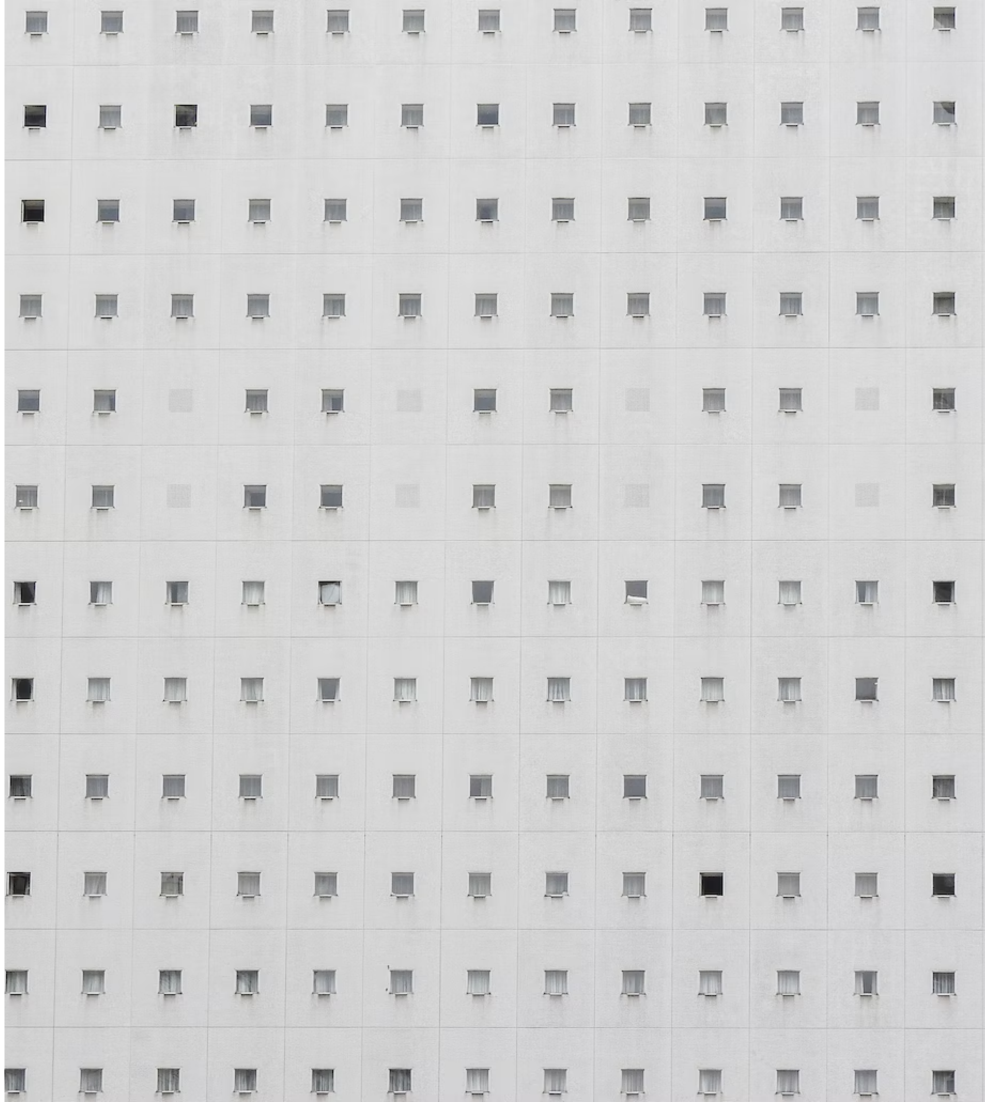
```
int result = add(1, 1);
// call the first function
```



## Background: Namespace

```
std::cout << "Hello, world!";
```

```
using namespace std;
// do not need `std::`
cout << "Hello, world!";
```



## `std::ranges` Namespace (C++20)

```
std::vector a = {1, 3, 2, 4};
```

with previous function...

```
// sort it
std::sort(a.begin(), a.end());
// it is sorted
assert(std::is_sorted(a.begin(), a.end()));
```

with `std::ranges`...

```
// sort it
std::ranges::sort(a);
// it is sorted
assert(std::ranges::is_sorted(a));
```

```
// std namespace
void std::sort(T first, T last);

// std::ranges namespace
void std::ranges::sort(T ranges);
void std::ranges::sort(I first, S last);
// we ignore the return value for simplicity
```

# Which Function will It Call?

```
// using namespace
using namespace std;
using namespace std::ranges;

vector a = {1, 3, 2, 4};
sort(a); // which function will it call?
```

```
// three candidates...
void std::sort(T first, T last);
void std::ranges::sort(T ranges);
void std::ranges::sort(I first, S last);
```

## Which Function will It Call?

```cpp
// using namespace
using namespace std;
using namespace std::ranges;

vector a = {1, 3, 2, 4};
sort(a); // which function will it call?
```

Compile Error!!

https://godbolt.org/z/s7bjT8xh1

```cpp
// three candidates...
void std::sort(T first, T last);
void std::ranges::sort(T ranges);
void std::ranges::sort(I first, S last);
```

## Argument-Dependent Lookup

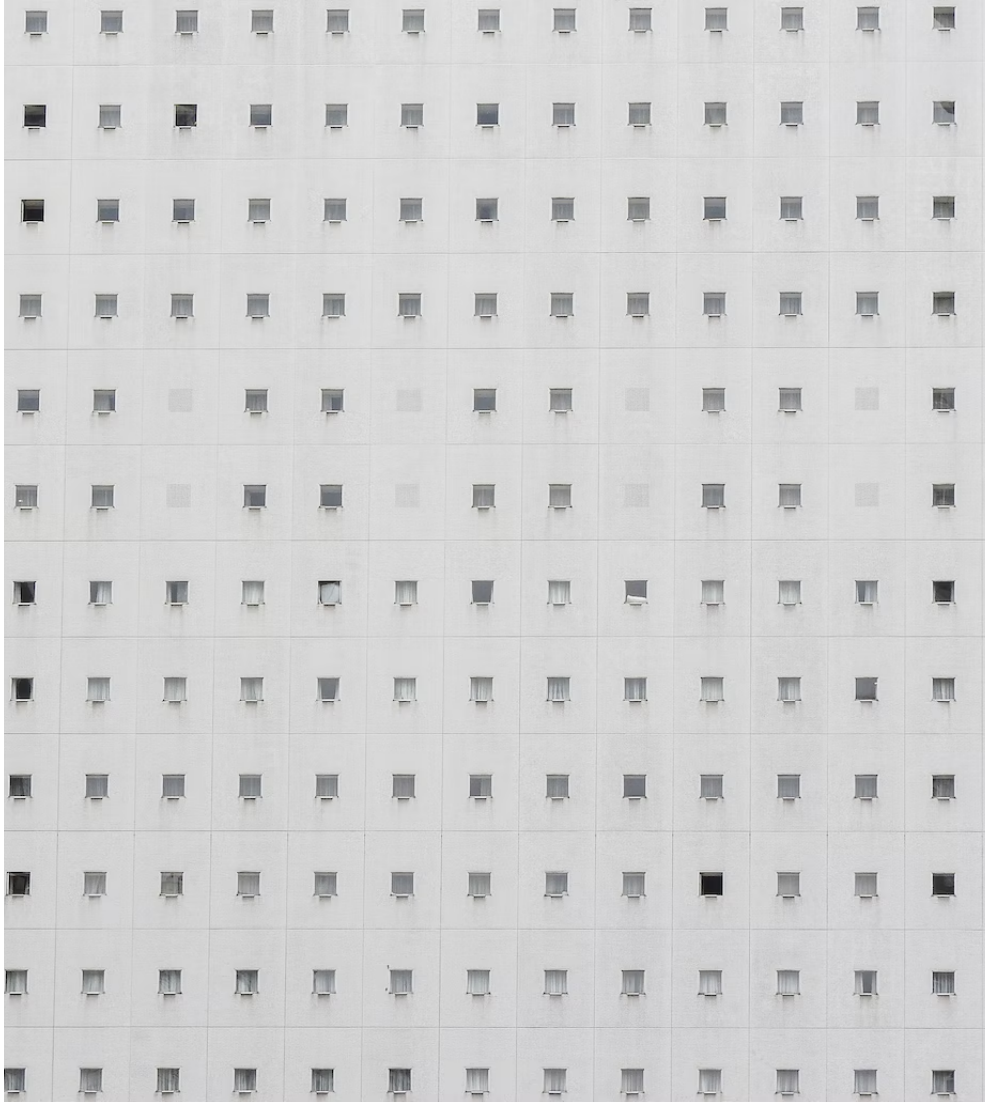
Recall the statement printing "hello world"...

```
std::cout << "Hello, world!";
```

is actually

```
operator<<(std::cout, "Hello, world!");
```

How does it find `` `std::operator<<` ``?



## Argument-Dependent Lookup

Recall the statement printing "hello world"...

```
std::cout << "Hello, world!";
```

is actually

```
operator<<(std::cout, "Hello, world!");
```

How does it find `std::operator<<`?

`std::cout` is inside `std`, the compiler will look for all possible ***functions*** in `std` as well

This allows us to write

```
std::cout << "Hello, world!"";
```

instead of

```
std::operator<<(std::cout, "Hello, world!");
```

## Here Comes the Problem...

```
using namespace std;

vector<int> a = {1, 3, 2, 4};
// which function will it call?
sort(a.begin(), a.end());
```

```
// three candidates...
void std::sort(T first, T last);
void std::ranges::sort(T ranges);
void std::ranges::sort(I first, S last);
```

```
using namespace std::ranges;
// not using std

std::vector<int> a = {1, 3, 2, 4};
// which function will it call?
sort(a.begin(), a.end());
```

## Here Comes the Problem...

```cpp
using namespace std;

vector<int> a = {1, 3, 2, 4};
// which function will it call?
sort(a.begin(), a.end());
```

```cpp
using namespace std::ranges;
// not using std

std::vector<int> a = {1, 3, 2, 4};
// which function will it call?
sort(a.begin(), a.end());
```

```cpp
// three candidates...
void std::sort(T first, T last);
void std::ranges::sort(T ranges);
void std::ranges::sort(I first, S last);
```

The second one still calls `` `std::sort` `` since we have ADL and `` `std::sort` `` is more suitable.

Counter-intuitive!

## Disable ADL: Niebloid!

If `` `sort` `` is a object instead of a function, ADL will not be effective.

Because ADL only work for actual functions.

```
namespace std::ranges {
struct __sort_fn {
    void operator()(R ranges) {
        // implementation
    }
};

// declare the functor object
__sort_fn sort;
} // namespace std::ranges

// call it
std::vector a = {1, 3, 2, 4};
std::ranges::sort(a);
// std::ranges::sort.operator()(a);
```

The functors that disable ADL are called Niebloids.

# Consequence…

Without ADL, we can call `` `std::ranges` `` correctly.

```cpp
using namespace std::ranges;
// not using std

std::vector<int> a = {1, 3, 2, 4};
sort(a.begin(), a.end());
```

Function overloading no longer works…

Here we have two identities…

```cpp
// using namespace
using namespace std;
using namespace std::ranges;

vector a = {1, 3, 2, 4};
sort(a);
```

`` `std::sort` `` is a function, while `` `std::ranges::sort` `` is an object.

Thanks

Q&A?